CSE 6512 Randomization in Computing
Exam I; Help Sheet

Basics: We started with the basic ideas behind randomized algorithms. Two kinds of randomized algorithms were defined, namely, Monte Carlo and Las Vegas. An array example of each kind was seen. The selection problem takes as input a sequence X of n elements and an integer $i, 1 \leq i \leq n$ and the problem is to find the ith smallest element of X. We showed that this can be solved using $n + \min\{i, n-i\} + \widetilde{o}(n)$ comparisons. The randomized sorting algorithm of Frazer and McKellar was discussed. This algorithm makes $n \log n + \widetilde{O}(n \log\log n)$ comparisons.

Fingerprinting: The technique of fingerprinting can be applied to check if two given objects are the same. Instead of comparing the objects directly, we apply a function on each and compare the images. Some examples we have seen are: 1) For three given $n \times n$ matrices A, B, and C, the problem is to check if $AB = C$. We pick a random vector r from $\{0,1\}^n$ and check if $A(Br) = Cr$. If so, we output: "$AB = C$"; else we output "$AB \neq C$". If $AB \neq C$, we showed that $Prob.[A(Br) = Cr] \leq 1/2$; 2) Given three polynomials $f(x), g(x)$, and $h(x)$, the problem is to check if $h(x) = f(x) \times g(x)$. The idea of fingerprinting can be used as follows: we pick a random element r from $\mathcal{S}$ (which is a subset of the field $\mathcal{F}$) and check if $f(r) \times g(r) = h(r)$. If so, we output: "$h(x) = f(x) \times g(x)$"; else we output: "$h(x) \neq f(x) \times g(x)$". We can show that the probability of an incorrect answer is no more than $\frac{d}{|\mathcal{S}|}$ where d is the degree of $h(x) - f(x) \times g(x)$; 3) Schwartz-Zippel theorem extends the above technique to check if a given multivariate polynomial is identically zero; 4) Edmonds' theorem in conjunction with Schwartz-Zippel theorem can be used to check for the existence of perfect matchings in graphs; 5) Karp-Rabin's algorithm applies fingerprinting to the string matching problem. To check if two given n-bit integers A and B are the same, the basic idea of this algorithm is to pick a random prime $p \leq t$ and check if $A \bmod p = B \bmod p$. Probability of an incorrect answer is $O\left(\frac{n}{t/(\log t)}\right)$.

Data structures: Random skip lists have been introduced. We have shown that the dictionary operations can be performed in an expected $O(\log n)$ time each, where n is the maximum number of elements in the data structure. We discussed hashing next. Let $M = \{0, 1, \ldots, m-1\}$ and $N = \{0, 1, \ldots, n-1\}$. A family H of hash functions from M into N is said to be 2-universal if for any two elements x and y from M with $x \neq y$, and for a randomly chosen h from H, $Prob.[h(x) = h(y)] \leq \frac{1}{n}$. We showed how to construct a family of 2-universal hash functions. Using this family, we discussed the data structure of AKS that can be used to search in $O(1)$ time in a static table.

Random walks on graphs: A random walk on a graph refers to starting from a node, moving to a random neighbor of that node, from there moving to a random neighbor, and so on. $C_u(G)$ refers to the expected length of a walk that starts at u and visits each node of G at least once. The cover time $C(G) = \max_u C_u(G)$. Any connected, undirected, and non-bipartite graph G induces a Markov chain M_G whose states are the vertices of G. For any two vertices $u, v \in V$, $P_{uv} = 1/d_u$ if $(u, v) \in E$; $P_{uv} = 0$ if $(u, v) \notin E$, d_u being the degree of u. Using this fact and known results on Markov chains, we showed that $C(G) \leq 2|E|(|V| - 1)$. We also stated the fact that $C(G) = O(|E|R(G)\log(|V|))$ where $R(G)$ is the resistance of the graph (when each edge in G is replaced with a resistance of one ohm).

The probabilistic method: Two ideas are prevalent: 1) Any random variable takes on a value that is at least as much as the mean; Also, it takes on a value that is at most the mean. 2) If the probability that a randomly chosen object from a universe satisfies a property P is positive, then there must be at least one object in the universe that satisfies P.

Using the above ideas we proved the following: 1) If $G(V, E)$ is any undirected graph, then there exists a partition of V into A and B such that the number of cross edges from A to B is $\geq |E|/2$;

2) For any set of m clauses there exists a truth assignment that satisfies at least $m/2$ clauses; 3) Let C_n be a complete graph on n vertices. Let $R(k,t)$ be the minimum value of n such that if the edges of C_n are colored with red and blue, then for each such coloring there exists either a red clique of size k or a blue clique of size t. If $\binom{n}{k}2^{1-\binom{k}{2}} < 1$, then $R(k,k) > n$; 4) There exists an $(n, 18, 1/3, 2)$ OR-concentrator, for all n larger than some constant; 5) A tournament on n nodes is a complete graph $G(V,E)$. Each node is a player. $< i,j > \in E$ if player i has defeated player j. We say that the tournament has property P_k if for every subset of k players there exists another player who has defeated all the k players. For every k there is a finite tournament with property P_k; 6) Let $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be a CNF Boolean formula where each clause has exactly k literals, k being some constant. Also, assume that each of the n variables occurs in at most $2^{k/10}$ clauses. Then, we used Lovász local lemma to show that the probability that a random assignment satisfies F is greater than zero.

Other problems: We have also studied the following problems: approximate matrix multiplication, min-cut, and primality testing.

PARALLEL ALGORITHMS. In a PRAM (Parallel Random Access Machine), processors communicate by writing into and reading from memory cells that are accessible to all. Depending on how read and write conflicts are resolved, there are variants of the PRAM. In an Exclusive Read Exclusive Write (EREW) PRAM, no concuurent reads or concurrent writes are permitted. In a Concurrent Read Exclusive Write (CREW) PRAM, concurrent reads are permitted but concurrent writes are prohibited. In a Concurrent Read Concurrent Write (CRCW) PRAM both concurrent reads and concurrent writes are allowed. Concurrent writes can be resolved in many ways. In a Common CRCW PRAM, concurrent writes are allowed only if the conflicting processors have the same message to write (into the same cell at the same time). In an Arbitrary CRCW PRAM, an arbitrary processor gets to write in cases of conflicts. In a Priority CRCW PRAM, write conflicts are resolved on the basis of priorities (assigned to the processors at the beginning).

We presented a Common CRCW PRAM algorithm for finding the Boolean AND of n given bits in $O(1)$ time. We used n processors. As a corollary we gave an algorithm for finding the minimum (or maximum) of n given numbers in $O(1)$ time using n^2 Common CRCW PRAM processors. The following result was also proven: The maximum of n arbitrary numbers can be found in $\widetilde{O}(1)$ time using n CRCW PRAM processors.

We also discussed a CREW PRAM algorithm for the prefix computation problem. This algorithm uses n processors and runs in $O(\log n)$ time on any input of n elements. (For the prefix computation problem the input is a sequence of elements from some domain Σ: $k_1, k_2, \ldots, k_n$ and the output is another sequence: $k_1, k_1 \oplus k_2, \ldots, k_1 \oplus k_2 \oplus k_3 \oplus \cdots \oplus k_n$, where $\oplus$ is any binary associative and unit-time computable operation on Σ.)

We also proved the following theorems: 1) Prefix computation on n elements can be done using $\frac{n}{\log n}$ CREW PRAM processors in $O(\log n)$ time; 2) If a parallel algorithm runs in time T on a P-processor PRAM, it can be simulated on a P'-processor PRAM in time $O(PT/P')$ as long as $P' \leq P$; 3) The selection problem on n elements can be solved in $\widetilde{O}(\log n)$ time using $\frac{n}{\log n}$ CREW PRAM processors; 4) We can sort n given arbitrary elements in $\widetilde{O}(\log n)$ time given n arbitrary CRCW PRAM processors; 5) We can sort n integers in the range $[1, (\log n)^c]$ in $O(\log n)$ time using $\frac{n}{\log n}$ CREW PRAM processors, c being any constant; 6) We can sort n integers in the range $[1, n(\log n)^c]$ in $\widetilde{O}(\log n)$ time using $\frac{n}{\log n}$ arbitrary CRCW PRAM processors, c being any constant; 7) We can sort n arbitrary elements in $\widetilde{O}\left(\frac{\log n}{\log\log n}\right)$ time using $n(\log n)^\epsilon$ arbitrary CRCW PRAM processors, ϵ being any constant > 0; and 8) We can sort n integers in the range $[1, n(\log n)^c]$ in $\widetilde{O}\left(\frac{\log n}{\log\log n}\right)$ time using $\frac{n(\log\log n)^2}{\log n}$ arbitrary CRCW PRAM processors.